Shared Control of Robot Manipulators With Obstacle Avoidance

A DEEP REINFORCEMENT LEARNING APPROACH

MATTEO RUBAGOTTI, BIANCA SANGIOVANNI, AIGERIM NURBAYEVA, GIAN PAOLO INCREMONA, ANTONELLA FERRARA and ALMAS SHINTEMIROV | Corresponding author: Antonella Ferrara, antonella.ferrara@unipv.it

INTRODUCTION

The word *teleoperation*, which in general means "working at a distance" is typically used in robotics when a human operator commands a remote agent. A teleoperated robot is often employed to substitute human beings in conditions where the latter cannot operate. A possible reason for it is the need to be in contact with dangerous substances, and indeed the first robot teleoperation system was designed in the 1940's for handling nuclear and chemical materials [1]. Other reasons can be the difficulty in bringing actual people in missions to explore deep waters or the space [2], [3], or the need to work at micro-scales, for example during a surgery [4], [5].

In certain cases, the reference provided by the human operator is not directly passed to the robot, but is instead used to generate an adaptive motion. This approach is known as *semi-autonomous teleoperation* or *shared control* [6], and its main aim is to reduce the workload of the human operator while this performs a difficult task that involves controlling a robotic system. The survey paper [7] provides an historical perspective of *bilateral teleoperation*, in which the controlled robot, which possesses force sensors, can transmit reaction forces from the task that is being performed back to the human operator via a haptic device (for the role of haptics in teleoperation and shared control, the reader is also referred to [8], [9] and the references therein). A review of control sharing methodologies is provided in [10] within the broader field of human-robot team interaction, highlighting aspects such as modeling of human behavior and human-machine interfaces. Shared control has been used in the past years in a number of different application domains, including unmanned aerial vehicles [11], walking-assistant robots [12], brain-actuated wheelchairs, [13], mobile robots [14], bimanual manipulation [15], human-robot interaction [16], robotic surgery [17], and snake robots [18].

A possible aim of shared control is that of automatically avoiding obstacles in the robot workspace. An example of such an application of shared control for robot manipulators is [19], in which a large-scale manipulator was controlled by mixing two velocity references, one provided by the operator and the other pre-planned by the control system, so as to "avoid risk-full actions" and to "adjust for variable or uncertain position of targets and obstacles" [19]. A different approach was presented in [20], where a shared control method based on artificial potential fields was used to avoid collisions of the teleoperated underwater manipulator Ocean One with obstacles such as rocks on

Digital Object Identifier 10.1109/MCS.2020.000000
Date of current version: XXXXXX

the sea bed. A different method for shared control relies on haptic force cues; in this case, a haptic device provides a force feedback aimed at preventing the operator from causing the robot to violate joint limits or to collide with obstacles (see, for instance, [21], [22] and the references therein cited).

The work presented in this paper considers a shared control problem to enforce obstacle avoidance for a teleoperated manipulator via deep reinforcement learning (DRL). The operator provides the position reference for the robot end effector, and the robot motion is generated in order to remain as close as possible to this reference, while satisfying joint limits and avoiding obstacles. Differently from [21], [22], the modification of the robot motion as compared to the reference happens without providing haptic feedback to the operator.

Thanks to the increased availability of computational power, and to the development of more efficient machine learning methods, DRL has seen a tremendous development in recent years [23]–[25]. In DRL, an agent (such as the software that controls a robotic manipulator) learns how to perform the best possible actions in a given environment, combining reinforcement learning (RL), which is based on mapping state-action pairs to corresponding expected rewards, with the powerful function approximation capabilities of deep neural networks (DNNs). In simple terms, a DNN acts as closed-loop controller, acquiring the system state as input and generating the action (control input) as its output. The DNN initially contains random weights. During the training phase, these weights are iteratively modified so that, when an action is generated, the given reward function is ideally maximized. In this way, the DNN "learns" to generate an optimal control law. DRL is thus a *data-driven (or model-free) method*, an approach in which the data are used directly to train the control law without using a process model, obtained either from first principles or via system identification. A more detailed overview of DRL, with specific focus on the Q-learning approach that will be used in this work, is provided in the Sidebar "Overview of Deep Reinforcement Learning". Among other fields, DRL has seen several interesting applications to the control of robot manipulators. As an example, the authors of [26] used a DRL algorithm to solve complex manipulation tasks, such as opening a door, without providing any prior demonstrations to the robot. The authors of [27] studied how to learn vision-based dynamic manipulation skills (in particular, grasping) by using a scalable DRL approach. In [28], two DRL algorithms were proposed and applied to a robot manipulator that executed real-world cloth manipulation tasks. Finally, in [29], a new model-based DRL algorithm was described, which focused on learning efficiently while imposing the satisfaction of complex constraints; the proposed method was then experimentally applied to a knot-tying task on a surgical robot.

DRL and teleoperation were already combined in [30], in which complex dexterous manipulation tasks were learned by a robot manipulator by using human demonstration to accelerate the convergence rate of a DRL algorithm. Also, in [31], an open-source framework was proposed for the control of robot manipulators based on state-of-the-art distributed reinforcement learning algorithms: among other features, the framework allowed the use of 3D motion devices to teleoperate the manipulators and collect human demonstrations. On the other hand, DRL has been used, outside the teleoperation domain, for the motion planning of robot manipulators with obstacle avoidance. For example, the authors of [32] studied the application of maximum entropy policies to robotic manipulation, including obstacle avoidance. In [33] and [34], collision avoidance problems for a robot manipulator in the presence of moving obstacles were solved via DRL. Also, the authors of [35] proposed a DRL-based nonprehensile rearrangement strategy for rearranging objects on a tabletop surface, including obstacle avoidance.

Summary

The recent surge of interest in the application of learning-based methods to control systems spurs this work to investigate how a purely model-free and a purely model-based method compare when applied to a shared control problem for a robot manipulator. Specifically, we propose a method based on model-free deep reinforcement learning (DRL) for tracking the position of an operator's hand with the end-effector of a manipulator, while automatically avoiding obstacles in the workspace with the whole robot frame. The obtained control strategy generates joint reference velocities via a deep neural network trained using Q-learning. The method is tested in simulation and experimentally on a UR5 manipulator, and compared with a model predictive control (MPC) approach for solving the same problem. It is observed that DRL presents a better performance than MPC, but only if the provided reference falls within the distribution of the DRL algorithm policy. As one can expect, the model-based nature of MPC allows it to cope with unforeseen situations, as long as these are compatible with its process model. This is not the case for DRL, for which an unexpected (not seen during the training process) human hand reference would lead to an extremely poor performance.

Despite the many opportunities that the use of data-driven methods presents, *model-based approaches* should also be taken into account, when possible, as viable solutions. For this reason, model predictive control (MPC) is here considered as an alternative approach. Similarly to DRL, MPC has also benefited from the increase in computational power that has taken place in the last

decades, but for different reasons. Indeed, with MPC, this power is used to solve an optimization problem at each sampling instant, rather than for offline computations aimed at training the control law, as it is the case with DRL. MPC is typically based on the minimization of a cost function (which is equivalent to maximizing rewards), and this constitutes a similarity with DRL. However, the reward maximization is not obtained via a learning process, but rather by minimizing the cost function based on the model-based time evolution of the system state over a given prediction horizon. This is achieved by solving an optimal control problem online, applying only the first of the computed control moves, and then re-solving the same optimal control problem based on the newly achieved state value [36]–[38]. The use of MPC has also been facilitated by the availability of various toolboxes such as ACADO [39], CVXGEN [40], and PANOC [41].

MPC has been applied to robot teleoperation in various scenarios, with the main aim of coping with communication delays [42], [43]. It has also been recently applied to various online motion planning for robot manipulators, for instance, in [44]–[46].

The difference between the design processes of DRL and MPC laws is shown in the flow diagram of Fig. 1. Both methods share their optimal control nature, since they can be both seen as approximate solutions of an ideal optimal control problem, and both will have to be tested first in simulation and then on the actual robot. However, on the one hand, MPC requires to define the mathematical models to predict the motions of robot and human before the controller is designed and tuned. On the other hand, DRL requires the definition of the hyperparameters that define the DNN (for example, number of layers and neurons per layer) and the learning algorithm, and the DNN will then learn to optimize the reward function without models of either robot or human motion.

The main innovative contribution of this work is the design and experimental testing of a DRL strategy, based on the model-free DRL approach proposed in [47], for teleoperating a robot manipulator while automatically avoiding collisions with obstacles in the robot workspace. Since an alternative method for solving the same type of problem using MPC was proposed in [48], a second contribution is the comparison of the proposed DRL strategy with that of [48], showing (based on simulation and experimental data) that it can provide improved performance under certain conditions. Our aim is to provide an answer, for robot shared control applications, to the more general question "To learn or not to learn?", which is a hot topic in the present control systems research. The considered case study is aimed at presenting an application example of advantages and disadvantages of DRL and MPC, which can be of interest for future applications.

DRL and MPC have already been compared (though only in simulation) in a few recent papers, which did not deal with robot manipulators, but rather with distributed wildfire surveillance control of a team of unmanned aerial vehicles [49], autonomous driving for merging into dense traffic [50], and adaptive cruise control [51]. These papers present some common conclusions with our work and some differences, which are discussed in the remainder of our paper.

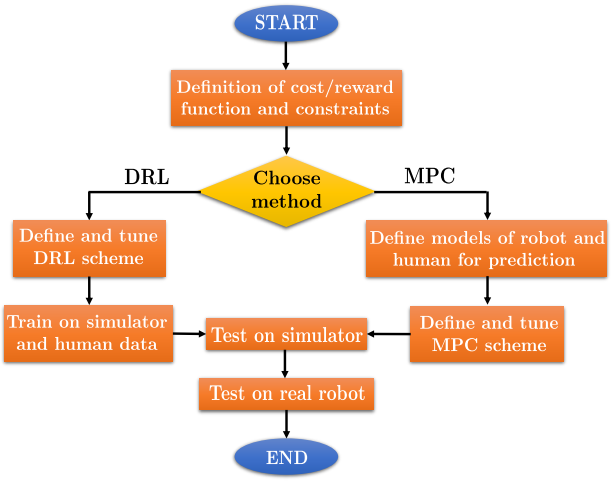

FIGURE 1 Flow diagram of the design processes of model predictive control (MPC) and deep reinforcement learning (DRL).

It is important to highlight that the shared control approach presented in this work can be seen as a particular case of a robot motion planning problem with obstacle avoidance. An overview of these problems is provided in the Sidebar "Motion Planning With Obstacle Avoidance".

DEEP REINFORCEMENT LEARNING APPROACH

In this section, some concepts of reinforcement learning (RL) are recalled, and recast in the considered robot teleoperation framework.

Background on reinforcement learning

As illustrated in Figure 2, the key idea of RL is that, having learned from past experience, it is possible to understand which *actions* lead to maximize a *reward function* in a given time horizon, for any given environment condition (*state*). In fact, the aim of RL problems is to maximize not only the current reward, but also the future ones, achieved as the result of actions influencing states. The element gathering information about states and performing actions trying to reach one or more goals related to the accomplishment of a predefined task is called *agent*. The design procedure for DRL in case of robot teleoperation is hereafter described, while further preliminaries on DRL are reported in the Sidebar "Overview of Deep Reinforcement Learning".

Overview of Deep Reinforcement Learning

MACHINE LEARNING

In computer science, machine learning (ML), as introduced in [S1], is the field that gives computers the ability to learn and accomplish a task without being explicitly programmed to do so, by the construction of algorithms that can learn from and make predictions on data. Thanks to increased computational power and availability of massive quantities of data, machine learning today finds application in many different domains, such as computer vision, natural language processing, identification, marketing analysis, and robotics. Its popularity is mostly due to its appealing property of being able to encode solutions for problems that are difficult to program explicitly. Furthermore, the same models can be adapted to different domains to solve other problems, with minimal hand-engineering. Nevertheless, one of the main drawbacks of machine learning is that it mostly works as a black box, meaning that, given its probabilistic nature, it is hard to interpret its results, which may lead to unexpected outcomes. Furthermore, the quality of a ML model greatly depends on the quality of the data used for training, which may not always be available: one of the most critical features of machine learning is indeed that of generalization, that is, the ability to perform with acceptable results even on data that were never explored during training, which is only possible if a proper data selection was made a-priori. Therefore, despite its promising potential, great care must be taken before applying machine learning to more critical domains.

In general, the goal of a ML model $\mathcal{M}$ is to represent the relation between some input data x and an output y, by means of a parametric function

$$\mathcal{M}_\theta(x) = y. \tag{S1}$$

The process of training is then the computation of the parameters θ of $\mathcal{M}$ in order to steer the output of the ML model as close as possible to the desired outcome. Usually, this happens through a *loss function*, used to quantify such divergence. ML can then be divided into three main categories. Specifically, the first one is the so-called "supervised learning", which is obtained by training the model using labeled data, meaning that, at each learning iteration, the output of the model is compared against the actual expected output, using some predefined metric; parameters are then updated and the procedure repeated until satisfactory results are obtained, and the model can correctly predict results for new data. The main applications of supervised learning are classification and data regression for model identification. The so-called "unsupervised learning", as the name suggests, is instead obtained by training the model with unlabeled data. Its main goal is to find common structures and repeated patterns in the input data, and it finds application in clustering, density estimation, and anomaly detection. Finally, "reinforcement learning" (RL) is a type of machine learning in which, differently from the previous two, usually no dataset is built a-priori, meaning that the model autonomously collects its data for training by interacting with the environment and observing the outcome of its action.

REINFORCEMENT LEARNING

RL [S2], is a branch of machine learning that, inspired by behavioral psychology, does not need a supervisor or a pre-built dataset in order to autonomously discover an optimal outcome for a specific task. The training process can be summarized as "learning by doing", and the results are achieved through iterative trial and error while interacting with an environment. Specifically, the RL framework relies on the concept that an *agent*, at any given time t, observes the environment, represented by a certain *state* $x_t \in \mathcal{X}$, with $\mathcal{X}$ being the *state space*, and according to a certain *policy* $\pi(u|x)$ performs a *certain action* $u_t \in \mathcal{U}$, with $\mathcal{U}$ being the action space, thus changing the state. When entering a new state, the agent receives a *reward* r_t, that is a scalar indicator on "how well" the agent has performed.

The framework is abstract and flexible: the *action space* and the *state space* are entirely arbitrary to the designer's discretion and can take a variety of forms with different degrees of complexity. The *policy* π, which serves as a mapping from states to action, can be either deterministic or probabilistic. For a given control task, the learning process is divided into episodes, where the agent interacts with the environment for either a complete attempt to perform a goal task or a fixed number of time-steps before being reset. The whole training process includes a typically large number of such episodes, up to a predefined maximum.

Reward

A *reward* r_t is a scalar feedback signal that indicates "how well" an agent has done at step t. The agent's goal is to maximize the (expected) cumulative reward it receives in the long run. In case of episodic tasks with finite horizon T, the expected cumulative reward R_t is defined as

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k+1}\,, \tag{S2}$$

where the term $0 \leq \gamma \leq 1$, known as *discount rate*, is used to prioritize earlier rewards over later ones: if γ is close to 0, the agent preferably chooses actions that maximize immediate rewards, while a value of γ close to 1 more likely results in the selection of a sequence of actions that leads to long-term maximization, despite possible immediate drawbacks. The reward function is perhaps the most crucial design element when setting a reinforcement learning framework, since it is the only form of "supervision" in the training process: therefore, it must be designed with great care in order to properly represent the desired behavior that the agent must accomplish.

Markov Decision Process

In order to better understand the RL principles, the concept of *Markov Decision Process* (MDP) needs to be introduced. Specifically, a MDP is defined by the state and action sets, and by the transition probability matrix of the environment. Given any state x and action u, the *transition probability* (or model) of the process is the distribution of each possible successive state x' and reward r, that is

$$p(x', r|x, u) = \mathbb{P}(x', r|x, u) \,, \tag{S3}$$

with p denoting the *dynamics* of the process and $\mathbb{P}(\cdot)$ being the probability function. Given any time t, a process is referred to as Markov process if and only if it satisfies the so-called Markov property

$$\mathbb{P}(x_{t+1}, r_{t+1}|x_t, u_t) = \mathbb{P}(x_{t+1}, r_{t+1}|x_t, u_t, x_{t-1}, u_{t-1}, ..., x_0, u_0) \,, \tag{S4}$$

meaning that the state at time $t+1$ depends only on the current state and action ("memorylessness" property). In RL, a task in which the transition probability of the environment satisfies the Markov property (S4) is called a MDP

$$< \mathcal{X}, \mathcal{U}, r, p, \gamma > \,. \tag{S5}$$

Therefore, given the current state x, action u and the next state x', the expected value of the reward is defined as

$$r(x, u, x') = \mathbb{E}\left[r_{i+1}|x_t, u_t, x_{t+1}\right] \,. \tag{S6}$$

Policy

The policy $\pi(u|x)$ defines the agent's behavior in the environment, and is the probability that the agent performs an action u while in state x. Depending on the task, the policy can be represented as a simple function or look-up table, or as a more complex deterministic or probabilistic function that requires extensive computation.

Value function

A *value function* V^π is an estimate of the value of moving into a certain state following an action provided by the policy π. Specifically, the value of a state is the expected cumulative reward the agent can gain over time, starting from that state and following the policy π thereafter, that is

$$V^\pi(x) = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|x_t = x\right] . \tag{S7}$$

$V^\pi(x)$ is the state-value function for the policy π. Furthermore, the value of a given state can be expressed in relation to the subsequent states by the so-called Bellman equation

$$V^\pi(x) = \mathbb{E}_\pi\left[R_t|x_t = x\right] \,. \tag{S8}$$

In a similar manner to the value function (S7), one can define the *action-value function* Q^π (also known as Q-function) as

$$Q^\pi(x, u) = \mathbb{E}_\pi\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1}|x_t = x, u_t = u\right] , \tag{S9}$$

representing the expected cumulative reward of taking action u while in state x, and following π thereon. Since the goal of reinforcement learning is to find an optimal policy that can maximize the cumulative reward, let us define π^* such that

$$\pi^*(x) = \text{argmax}_\pi \, V^\pi \qquad \forall x \in \mathcal{X}. \tag{S10}$$

Therefore, one can define the *optimal value function* as

$$V^*(x) = \max_\pi V^\pi(x) \,, \tag{S11}$$

and, recalling the expression of the action-value function, one can express the Bellman equation for V^* as

$$V^\pi(x) = \max_{u \in \mathcal{U}} Q^{\pi^*}(x, u) \,. \tag{S12}$$

Intuitively, the Bellman optimality equation for Q^* is obtained by substituting $V^*(x')$ with $Q^*(x', u')$. This equation expresses that, under an optimal policy, the value of the state is equal to the expected reward obtained after performing the best action from thereon. Furthermore, since (S12) is a system of N equations in N unknowns, given an environment with known dynamics, one can solve the value of V^* for each state, and then determine the optimal policy π^* to solve a task. However, the model of the environment is not always available upfront.

Q-learning

When dealing with complex systems, such as those found in robotics applications, the complete transition probability of the environment may not be available, requiring the use of model-free learning approaches. One way to achieve this is through Q-learning, introduced in [S3], which enables the direct approximation of the optimal action-value function Q^* independently of the policy being applied. Specifically, the algorithm performs a step-wise update of the approximator $\hat{Q}$, by computing a temporal difference between two consecutive values and updating the value of $\hat{Q}$ according to a learning rate $\alpha \in [0, 1]$. Furthermore, under the assumption of performing each action infinitely often, and visiting each state infinitely often, with $\alpha \to 0$, then

$$\hat{Q}(x, u) \to Q^*(x, u) \tag{S13}$$

with probability 1.

Now, let us introduce the concepts of "exploration and "exploitation" in reinforcement learning. Exploration during training means that the agent tends to transition to other states regardless of policy optimization. Exploration is usually achieved by either randomizing actions entirely or by adding noise to the selected action u. This enables the agent to explore states that would otherwise risk never getting evaluated, thus preventing finding a possible better policy. In case of exploitation, on the contrary, the agent uses already acquired data in order to make decisions. Although this may lead to finding a policy faster, it usually results in suboptimal outcomes, since the agent does not know of other possibly better solutions. A policy that only uses exploitation is called "greedy". Typically, one design element in a reinforcement learning framework is the balance between exploration and exploitation, in order to ensure fast convergence and fair coverage of all actions, states, and rewards of the environment.

Hyperparameters

Finally, in ML there are predefined parameters, called hyperparameters, whose values are not learned during the training process but rather decided upfront during the design phase. In practice, these values are determined through a preliminary search activity aiming to identify the most effective combination.

DEEP REINFORCEMENT LEARNING

In case of problems with an overwhelmingly large (possibly infinite) number of possible states, one way to overcome the issue of mapping the relationships between states and actions is to use Deep Neural Networks (DNNs) to build a parametric approximator of the Q-function. The most notable example of DNNs being used to solve reinforcement learning problems is the work [52], where superhuman performances were achieved when training an agent to play Atari videogames. DNNs are powerful parametric functions capable of modeling complex nonlinear relationships between inputs and outputs by means of neurons, to which certain weights w are assigned, connected in k layers; the term "deep" refers to the level of composition of the parameters and the use of multiple "hidden" layers between the input and output ones. Let us assume to have a generic target function

$$y = f(x), \quad x \in \mathbb{R}^n \,. \tag{S14}$$

Then, a parametric representation of the function with a DNN is expressed as

$$\begin{aligned} \hat{y} = W \cdot g(W^{(1)} \cdot g(W^{(2)} \cdot \\ g(...g(W^{(k)} \cdot x + b^{(k)})...) + b^{(2)}) + b^{(1)}) + b \,, \end{aligned} \tag{S15}$$

where $W^{(k)}$ is the set of weights associated with neurons of the k-th layer, g is an *activation function*, and $b^{(k)}$ is the set of biases associated with the output of the k-th layer. Generally speaking, the aim of training a DNN is that of updating the set of parameters θ (that is the weights and bias) so that the output of the network matches the desired one.

Making now reference to Q-learning, let us consider a parametrized Q-function $\hat{Q}(x, u|\theta^Q)$. Then, let us impose a stochastic behavior policy β such that $u_t = \beta(x_t)$, with a state visitation frequency ρ^β (that is the number of times a state is visited using stochastic policy β). Under these assumptions, the goal is now to minimize a *loss function*

$$L(\theta^Q) = E\left[\left(\hat{Q}(x_t, u_t|\theta^Q) - y_t\right)^2\right] \,, \tag{S16}$$

where the term y_t is the target function such that

$$y_t = r(x_t, u_t) + \gamma \hat{Q}(x_{t+1}, \hat{\mu}(x_{t+1})|\theta^Q) \tag{S17}$$

and

$$\hat{\mu}(x_t) = \text{argmax}_{u_t}\, \hat{Q}(x_t, u_t|\theta^Q). \tag{S18}$$

Several algorithms have been proposed with promising results for solving (S18), such as deep deterministic policy gradients (DDPG) [S5], twin delayed DDPG (T3D) [S6], and soft actor-critic [S7]. Nevertheless, when dealing with a very large number of continuous states and actions, it is especially useful to be able to rephrase the optimization problem so that the computation of the policy can be performed more efficiently. This can be achieved by the Normalized Advantage Function (NAF) algorithm, introduced in [47].

Normalized Advantage Function

Making reference to the Q-learning problem as in [47] (see also [53] for further theoretical details), let us consider a parameterized Q-function $\hat{Q}\left(x_t, u_t|\theta^Q\right)$ such that

$$\hat{Q}\left(x, u|\theta^Q\right) = \hat{A}\left(x, u|\theta^A\right) + \hat{V}\left(x|\theta^V\right) \,, \tag{S19}$$

where $\hat{A}$ and $\hat{V}$ are DNN approximators of the so-called *advantage function* $A^\pi(x_t, u_t) = Q^\pi(x_t, u_t) - V^\pi(x_t)$ and the value function $V^\pi(x_t)$, respectively. Specifically, the advantage term is expressed as the quadratic function

$$\hat{A}\left(x, u|\theta^A\right) = -\frac{1}{2}(u - \hat{\mu}(x|\theta^\mu)^\top P(x|\theta^P)(u - \hat{\mu}(x|\theta^\mu)) \,. \tag{S20}$$

The function $P(x|\theta^P)$ is a positive-definite square matrix defined such that

$$P(x|\theta^P) = L(x|\theta^P)L(x|\theta^P)^\top \,, \tag{S21}$$

with $L(x)$ lower triangular and with entries derived from the outer layer of the dedicated neural network. The policy (S18) can then be computed as

$$\frac{\partial}{\partial u}\hat{Q}\left(x, u|\theta^Q\right) = -(u - \hat{\mu}(x|\theta^\mu))^\top P(x|\theta^P) \,. \tag{S22}$$

By imposing equality to 0, one gets

$$\frac{\partial}{\partial u}\hat{Q}\left(x, u|\theta^Q\right) = 0 \,, \tag{S23}$$

that is $u = \hat{\mu}(x|\theta^\mu)$. Therefore, the action maximizing the Q-function is always $u = \hat{\mu}(x|\theta^\mu)$.

REFERENCES

[S1] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of Research and Development*, vol. 44, no. 1.2, pp. 206–226, 1959.

[S2] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*, vol. 2, Cambridge: MIT Press, 1998.

[S3] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[S4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[S5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[S6] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning (ICML)*, vol. 80, (Stockholmsmässan, Stockholm, Sweden), pp. 1582–1591, PMLR, 10–15 Jul 2018.

[S7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning (ICML)* (J. Dy and A. Krause, eds.), vol. 80, (Stockholmsmässan, Stockholm Sweden), pp. 1861–1870, PMLR, 10–15 Jul 2018.

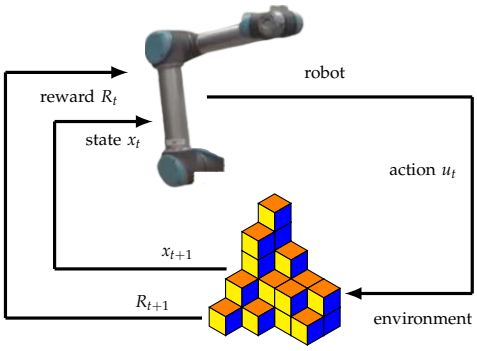

FIGURE 2 Block diagram of robot/environment interaction concept behind reinforcement learning (RL).

Deep reinforcement learning robot setting

The proposed framework is now formulated making reference to Figure 2, with $x_t \in \mathcal{X}$ being the state at time t, and the *state space* $\mathcal{X}$ given by

$$\mathcal{X} \triangleq \{q, p_e, p_d, \mathcal{O}\} \ , \tag{24}$$

where $q \in \mathbb{R}^{n_q}$ is the vector of measured robot joint positions, $p_e \in \mathbb{R}^3$ is the end-effector position, $p_d \in \mathbb{R}^3$ is the corresponding reference position, and finally $\mathcal{O} \in \mathbb{R}^{n_\mathcal{O}}$ represents the information about the position of one or more obstacles in the robot workspace. The value of $n_\mathcal{O}$ represents the number of scalars needed to store a representation of the obstacle (for instance, the coordinates of its vertices for a polygonal obstacle). Let the vector of reference joint velocities at each joint $\dot{q}_d \in \mathbb{R}^{n_q}$ correspond to the action u_t performed by the robot given a certain state at time t, and let $\mathcal{U}$ be the action space, that is

$$\mathcal{U} \triangleq \{\dot{q}_d\} \ . \tag{25}$$

An action issued at time $t-1$ influences the state space starting from time t. Thus, a transition to state x_t triggered by an action u_{t-1} yields a *reward* r that is a scalar feedback signal expressing "how well" the robot has done at step t. In the considered case, the reward function is given by

$$r(x_t, u_{t-1}) \triangleq -c_1 r_T - c_2 r_U - c_3 r_\mathcal{O} \ , \tag{26}$$

in which $c_1, c_2, c_3 \in \mathbb{R}_{>0}$ are tuning parameters, while the meaning of r_T, r_U, and $r_\mathcal{O}$ is explained in the following. The term r_T in (26) is related to the distance between the end effector and the reference trajectory, and is computed relying on the so-called Huber-loss function, that is

$$r_T \triangleq \begin{cases} \frac{1}{2} d^2 & \text{for } d < \delta_T \\ \delta_T \left(d - \frac{1}{2} \delta_T \right) & \text{otherwise} \end{cases} \ , \tag{27}$$

where d is the Euclidean distance between the end-effector and the reference, while δ_T is a threshold that determines the transition between the quadratic and the linear parts of r_T. The second term r_U is a regularization term, computed as

$$r_U \triangleq \|u\|^2 \ , \tag{28}$$

which has the purpose of encouraging smaller control actions. Finally, $r_\mathcal{O}$ defines the penalty for being too close to obstacles, and is computed as

$$r_O \triangleq \left(\frac{\delta_\mathcal{O}}{\delta_\mathcal{O} + d_\mathcal{O}} \right)^g \ , \tag{29}$$

where $d_\mathcal{O}$ is the minimum distance between the robot frame and the obstacles, while $\delta_\mathcal{O} \in \mathbb{R}_{>0}$ is a constant parameter ensuring that $0 < r_O < 1$, and $g \in \mathbb{R}_{>0}$ is a hyperparameter aimed at increasing the decay rate of the reward component when the robot is far from the obstacles. Since the DRL algorithm is usually trained in simulation, the value of $d_\mathcal{O}$ can typically be extracted from the specific simulator without the need to know the algorithm used for computing it (see, for instance, [54]). Notice that $r_\mathcal{O}(d_\mathcal{O})$ is a monotonically decreasing function, with $r_\mathcal{O}(0) = 1$, and $\lim_{d_\mathcal{O} \to \infty} r_\mathcal{O}(d_\mathcal{O}) = 0$.

For episodic tasks involving a sequence of n actions, we define the *cumulative reward* as

$$R_n \triangleq \sum_{k=0}^{n} \gamma^k r_{t+k+1} \ , \tag{30}$$

where the term $0 \leq \gamma \leq 1$ is the *discount rate*, used to prioritize earlier costs over later ones.

Considered continuous Q-learning approach

A Q-learning approach is adopted in this paper. This relies on the definition of the so-called *action-value function*, or *Q-function*, namely $Q^\pi(x, u)$, corresponding to the *policy* $\pi : \mathcal{X} \mapsto \mathcal{U}$. The aim of Q-learning is to determine the *optimal policy* π^*, that is the one that maximizes the expected cumulative cost, starting from any given state. To do this, we use a DNN as parametric approximator of the Q-function, and we refer to this approximation as $\tilde{Q}$. We introduce the *value function* $V^\pi : \mathcal{X} \mapsto \mathbb{R}$, defined as the expected cumulative reward accumulated by the robot by adopting π from a given state on, and the *advantage function* $A^\pi(x_t, u_t) \triangleq Q^\pi(x_t, u_t) - V^\pi(x_t)$. The DNN defining $\tilde{Q}$ is trained by incrementally updating its parameters, and ideally converging to the optimal Q-function. The approximator is chosen for instance as

$$\tilde{Q}\left(x, u | \theta^Q\right) \triangleq \tilde{A}\left(x, u | \theta^A\right) + \tilde{V}\left(x | \theta^V\right) \ , \tag{31}$$

where $\tilde{A}$ and $\tilde{V}$ are parametric approximators of A^π and V^π, respectively, and θ^A and θ^V the corresponding vectors of parameters. The aim of the learning process thus becomes that of learning a greedy deterministic policy $\pi(u_t | x_t) = \delta(u_t = \tilde{\mu}(x_t))$, with $\tilde{\mu}(x_t) = \text{argmax}_u \tilde{Q}(x_t, u_t)$, by minimizing a *loss function* given by

$$L(\theta^Q) \triangleq \mathbb{E}_{r, \rho^\beta, \beta}\left[\left(\tilde{Q}\left(x_t, u_t | \theta^Q\right) - y_t \right)^2 \right] \ , \tag{32}$$

where the term y_t is the target

$$y_t = r(x_t, u_t) + \gamma \tilde{Q}\left(x_{t+1}, \tilde{\mu}(x_{t+1}) | \theta^Q\right) \ , \tag{33}$$

Our aim is to provide an answer, for robot shared control applications, to the more general question "To learn or not to learn?"

with θ^Q containing the parameters of the action-value function approximation, β being a stochastic behavior policy such that $u_t = \beta(x_t)$, and ρ^β being the state visitation frequency with policy β.

Due to the continuous nature of the robot teleoperation problem, the approach based on the so-called *normalized advantage function* (NAF) [47] is finally used to find the optimal policy [53]. The conceptual idea is that NAF adopts a quadratic Q-function approximator such that $\text{argmax}_u \tilde{Q}(x_t, u_t)$ can be efficiently computed during each update. More specifically, the algorithm uses a DNN to separately output the terms $\tilde{A}$ and $\tilde{V}$, where the advantage term is parametrized as a quadratic function, that is

$$\tilde{A}\left(x, u|\theta^A\right) = -\frac{1}{2}\left(u - \tilde{\mu}(x|\theta^\mu)\right)' P(x|\theta^P)\left(u - \tilde{\mu}(x|\theta^\mu)\right) , \tag{34}$$

in which the term $P\left(x|\theta^P\right)$ is a state dependent, positive-definite square matrix. The Q-function is quadratic in the action and, by computing the maximum, one has

$$\frac{\partial}{\partial u}\tilde{Q}(x, u|\theta^Q) = \left(\tilde{\mu}(x|\theta^\mu) - u\right)' P(x|\theta^P) = 0 . \tag{35}$$

Therefore, the action u that maximizes $Q(x, u)$ is always given by $u = \tilde{\mu}(x|\theta^\mu)$.

Note that, apart from the weights, the tuning of the proposed DRL is performed via a set of hyperparameters, which are defined upfront during the design phase. The hyperparameters involved in the performance of the experiments are summarized for the specific case study considered in this paper in the following sections. Furthermore, in order to achieve the best policy, extensive training needs to be performed. Typically and in practice, any such training process includes a large number of episodes, up to a predefined maximum. During each episode the robot interacts with the environment for either a complete attempt to perform a goal task or a fixed number of time-steps before being reset. Finally, after training, the performance is evaluated in terms of a cumulative reward function computed over reference trajectories that were not considered during the training phase.

MODEL PREDICTIVE CONTROL APPROACH

This section summarizes a variation of the MPC approach described in [48] for solving the same teleoperation problem here considered, by using the RL jargon (for example, reward functions are used instead of cost functions) in order to highlight the commonalities with the proposed DRL approach. To approximate the optimal policy π^*, instead of defining a deterministic policy $\tilde{\mu}(x|\theta^\mu)$ based on machine learning, a finite-horizon optimal control problem (FHOCP) is solved online to directly determine a different deterministic policy $\hat{\mu}(x|\theta^{MPC})$, where θ^{MPC} is the set of MPC tuning parameters, as detailed in the remainder of this section. At any given time instant t, the reward function is defined as

$$\hat{r}(x_t, u_t) \triangleq -c_1\hat{r}_T - c_2 r_U , \tag{36}$$

in which $\hat{r}_T$ is a purely quadratic function (needed to be able to efficiently determine $\hat{\mu}(x|\theta^{MPC})$ online), defined as $\hat{r}_T \triangleq \frac{1}{2}d^2$. Notice that the penalty for obstacle collisions is not inserted in the reward function, as an explicit constraint on obstacle avoidance can be instead directly inserted in the FHOCP. The cumulative reward for MPC is thus defined as

$$\hat{R}_t \triangleq \sum_{k=0}^{N-1} \hat{r}_{t+k+1|t} , \tag{37}$$

where $N \in \mathbb{R}_{>0}$ is the so-called *prediction horizon*, while $\hat{r}_{t+k+1|t}$ is the forecast of $\hat{r}$ at time $t+k+1$, predicted at time t. The FHOCP aims at determining the optimal realization of the control sequence

$$\boldsymbol{u}_t \triangleq \{u_{t|t}, u_{t+1|t}, \ldots, u_{t+N-1|t}\} ,$$

namely $\boldsymbol{u}_t^*$, based on a prediction of the evolution of the obstacle configuration

$$\boldsymbol{O}_t \triangleq \left\{\mathcal{O}_{t+1|t}, \mathcal{O}_{t+2|t}, \ldots, \mathcal{O}_{t+N|t}\right\}$$

and of the reference

$$\boldsymbol{p}_{d,t} \triangleq \left\{p_{d,t+1|t}, p_{d,t+2|t}, \ldots, p_{d,t+N|t}\right\}$$

from the human operator, both predefined at time t. In the considered case study, the obstacles are assumed to maintain a given constant position. Also, in order to obtain a completely model-based MPC law, and independent from any learning data, the prediction of the reference value $\boldsymbol{p}_{d,t}$ is obtained extrapolating the hand motion based on the current direction and speed, both estimated from the last measured samples. In order to generate the value of $\boldsymbol{p}_{d,t}$ for the whole prediction horizon, it is assumed that the hand motion will be rectilinear (maintaining the current direction) and with a constant speed (equal to the currently observed one). On the other hand, the prediction of the robot configuration, namely

$$\boldsymbol{q}_t \triangleq \left\{q_{t|t}, q_{t+1|t}, \ldots, q_{t+N|t}\right\} ,$$

is determined by the predicted realization of the control sequence. We can finally provide the FHOCP formulation

$$\boldsymbol{u}_t^* = \text{argmax}_{q_t, \boldsymbol{O}_t, \boldsymbol{p}_{d,t}, \boldsymbol{u}_t} \ \hat{R}_t(q_t, \boldsymbol{p}_{d,t}, \boldsymbol{u}_t) \tag{38a}$$

$$\text{s.t. } q_{t|t} = q_t \tag{38b}$$

$$q_{t+k+1|t} = q_{t+k|t} + \tau_s u_{t+k|t}, \ k = 0, \ldots, N-1, \tag{38c}$$

$$\hat{d}_{\mathcal{O}}(q_{t+k|t}, \mathcal{O}_{t+k|t}) \geq 0, \ k = 1, \ldots, N \ , \tag{38d}$$

where (38b) imposes that the sequence of predicted joint variables $q_{t+k|t}$ starts from the currently measured value q_t, (38c) determines a simplified system dynamics (by assuming that the desired joint velocity, instead of the actual joint velocity, directly influences the joint position) where $\tau_s \in \mathbb{R}_{>0}$ is the sampling interval, while (38d) imposes obstacle avoidance. More specifically, $\hat{d}_{\mathcal{O}}(q, \mathcal{O})$ represents a conservative estimate of the distance between robot and obstacles based on an over-approximation of their space occupancy. Indeed, contrary to the calculation of the distance $d_{\mathcal{O}}$ used in DRL, which can be done through a simulator, an explicit formula is needed to insert this distance inside the FHOCP. As an example, test points can be defined on the robot frame and on the obstacles, each of them being at the center of a sphere with given radius; if the union of the spheres on the robot includes the whole robot frame, and analogously for the obstacle, then condition (38d) can be imposed by requesting that all distances between each test point on the robot and each test point on the obstacle are greater or equal than the sum of the corresponding sphere radii.

Once the FHOCP (38a)-(38d) is solved at time t, only the first element of the control sequence determines the MPC policy $\hat{\mu}(x|\theta^{MPC}) = u_{t|t}^*$, and the FHOCP is then solved at time $t+1$, after new measurements are available. The set of MPC parameters θ^{MPC} includes c_1 and c_2, the parameters of the method used to formulate (38d), and those of the method used to predict future obstacle and reference positions.

MAIN DIFFERENCES BETWEEN DRL AND MPC

Compared to DRL, the MPC approach presents the following main differences. First, MPC aims at tracking the reference while avoiding the obstacles based on an explicit model-based prediction of the robot motion on a limited time horizon, while DRL learns its behavior via trial and error (during training). Second, MPC is based on a simplified system dynamics in order to limit the computational complexity of the FHOCP, while DRL can be trained on a sophisticated robot simulator, which replicates the dynamics of the manipulator and the behavior of its internal controllers. Third, MPC uses a conservative evaluation of the distance with the obstacles (again, to limit computational complexity), while DRL can rely on precise distances calculated by the employed simulator during training (although the presence of the penalty term $-c_3 r_O$ in the reward function, rather than a hard constraint as in MPC, can introduce some conservativity in DRL too). Fourth, MPC does not need training data, and is therefore much more robust towards managing unforeseen events, such as a reference motion never seen before; DRL might instead show an unexpected behavior should this happen. Fifth, the DRL training might fail for certain values of the hyperparameters. More in general, teleoperation is often used to work in unstructured or safety-critical remote environments, where machine learning algorithms tuning (for instance, weights, hyperparameters) or extensive training (especially with a large number of episodes) and evaluation are not easily carried out. In such environments, if the DRL training fails, then model-based approaches such as MPC or methods based on haptic force cues such as [21], [22] would constitute a better solution.

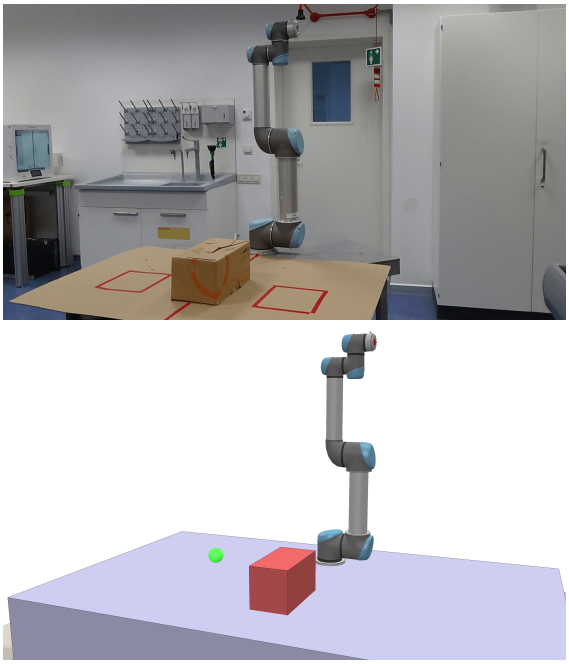

FIGURE 3 Experimental setup used in the case study. The setup consists of a UR5 manipulator and obstacles (top), while the virtual reconstruction of the same setup is done in the CoppeliaSim simulator (bottom).

CASE STUDY

The scenario on which the proposed methodologies were applied involved an UR5 6-axis robot manipulator tasked with the tracking, with its end-effector, of a reference position generated online by an operator; while doing so, the robot had to avoid collisions with a box placed on a table, and with the table itself. The considered scenario, including the obstacles, is represented in Figure 3 (top).

The operator's hand motion was converted into a suitable reference signal p_d for the robot via a custom-made 7-

DOF human arm motion tracker system [55]. The recorded motions consisted of movements of the arm from the left to the right of the operator, and vice versa, while varying the height of the hand. In the robot reference frame $O - xyz$, the values of the xyz components of p_d varied, respectively, within $[-0.57, +0.53] \times [-0.15, +0.60] \times [-0.23, +0.72]$ m. Most of the trajectories generated for either training or testing were such that, if the robot had to perfectly follow the provided end-effector reference, a collision with the obstacles would happen at a certain point in time.

Deep reinforcement learning design and training

The training of the DRL agent was performed in simulation by interfacing the NAF algorithm with the CoppeliaSim simulator (see the bottom part of Figure 3), encapsulating the environment, via the PyRep [56] plugin. Synchronous simulations were run with a sampling interval $\tau_s = 50$ ms and had a duration of 18 s (360 time steps), in which the simulated UR5 received values of p_d from the training data set recorded with the above-mentioned arm motion tracking system, performed the actions provided by the policy, and observed the environment as it changed, collecting experience. The employed trajectories were sampled from the training set (consisting of a total of 447500 data points) at the beginning of each episode. Before evolving with time, the reference idled for 5 seconds in order to allow the robot to reach the selected starting point of the randomly chosen trajectory segment. After each episode, both the trajectory and its starting point were reset, and the robot was re-initialized at its home configuration.

It is worth noticing that, according to (24), the currently measured obstacle position should in general be an input for the DRL control law. As a consequence, it was decided to keep providing the coordinates of the obstacle center to the DRL controller, even if, given the fact that in the considered scenario the obstacle was static, this was redundant information. However, the same DNN can be in principle trained also for scenarios in which the given obstacle changes its position with time.

The training was executed for 700 episodes: the corresponding learning curve is reported in Figure 4 (red line), in which we can notice that, for episodes with duration of 18 s, the total reward J_n, evaluated a posteriori without discount factor, that is

$$J_n \triangleq \sum_{t=1}^{n} r_t \,, \tag{39}$$

appears to converge to a value of -10^4. Making reference to the DRL framework and the NAF algorithm as discussed in [47], the hyperparameters used during the training phase are summarized in Table 1.

After completing the training phase, the DRL algorithm was tested in simulation using the dynamics model also

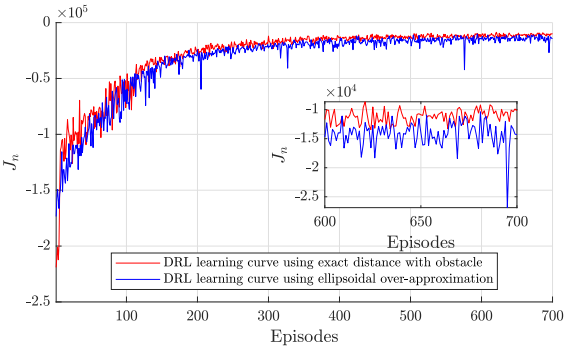

FIGURE 4 Deep reinforcement learning (DRL) learning curves for the UR5 case study.

TABLE 1 Deep reinforcement learning (DRL) hyperparameter values.

Parameters	Value
number of time steps	360
time step	50 ms
c_1	1000
c_2	200
c_3	60
δ_T	0.1
g	8
δ_O	0.1
discount factor γ	0.99
update factor τ	0.001
learning rate	0.001
noise type $\mathcal{N}$	*Ornstein–Uhlenbeck*
noise decay factor	0.01
noise scale	1

used for training and implemented in CoppeliaSim, on a test set consisting of 500 new reference trajectories, spanning the same range of values used during training, each of them with a total duration of 40 s (that is 800 steps). As a result, an average value of $J_n = -1.69 \cdot 10^4$ was obtained, with a standard deviation of $2.70 \cdot 10^3$. In order to compare these results with those represented in the training curve, the same metrics have been evaluated in the first 18 s of the simulations in the test set, obtaining an average value of $J_n = -8.26 \cdot 10^3$, with a standard deviation of $1.10 \cdot 10^3$. The result is in line with the final value of $J_n = -10^4$ shown in the learning curve, even though it presents a slight improvement. Also, no collisions were observed between the robot and the obstacles for this test set.

As already mentioned when explaining the main differences between the DRL and MPC approaches, one advantage of DRL is the ability of using precise distances between robot and obstacle, obtained during the training phase using the employed simulator. MPC, instead, has to rely on a conservative evaluation of such distance: more precisely, in the MPC approach that will be explained in the following, the space occupied by the box will be over-approximated by an ellipsoid, which will force the robot to move farther from the actual obstacle. In order to show that

this constitutes an additional layer of conservativity for MPC (partially compensated by the fact that MPC can impose hard constraints on obstacle avoidance in the FHOCP (38)), we trained the same DRL algorithm described above so as to avoid the over-approximating ellipsoid rather than the box itself. The result in terms of learning curve can be observed in Figure 4 (blue line), where one can see that the newly-defined DRL algorithm converges to a worst value than the original DRL case. Also, the new algorithm was tested on the same set of 500 reference trajectories used for testing the original DRL case, obtaining an average value of $J_n = -2.34 \cdot 10^4$, with a standard deviation of $3 \cdot 10^3$. As the original DRL algorithm had, as expected, a better performance, the newly-defined DRL algorithm will not be considered in the remainder of the paper.

Model predictive control design

In order to provide a comparison, an MPC strategy was designed using, to define the reward in (36), the same parameters c_1 and c_2 as in Table 1, and a prediction horizon $N = 20$, corresponding to a total prediction time of 1 s. As already mentioned above, the prediction of the end-effector position reference $\boldsymbol{p}_d$ was obtained by linearly extrapolating the current value of p_d based on an estimate of its current speed $\dot{p}_d$, as in [48]. Also, the avoidance of box and table was obtained similarly to the description provided before, by defining 7 test points on the manipulator, and the same number of spheres that cover the whole robot frame (this is a less conservative approach as compared to [48], in which only 3 spheres, with larger radii, were defined on the robot). The space occupied by the box was covered by the above-mentioned ellipsoid, and the condition for imposing absence of intersections between the spheres on the robot and the ellipsoid was imposed as described in [48, Sec. II.C]. Finally, the avoidance of the table was also imposed as described in [48, Sec. II.C], by requiring that all spheres on the robot remained entirely above the horizontal plane that defined the table surface.

EXPERIMENTAL RESULTS AND DISCUSSION

In this section, the experimental results for the considered UR5 case study are presented and discussed. Both controllers were implemented in Robot Operating System (ROS) and ran on an Acer laptop with a 2.6GHz Intel Core i7-9750H CPU with 16GB RAM. The UR5 robot provided, every 50 ms, joint positions and position of the end effector to the controllers via TCP/IP communication, and the controllers transmitted the control inputs (coinciding with the joint speed references for the internal control loops) over the same connection as URScript language commands. The optimization solver for MPC was generated using the ACADO Toolkit [39], [58], [59], which includes a code generation tool. The FHOCP (38) was formulated via multiple shooting with a discretization interval coinciding with the sampling interval of 50 ms. A Gauss-Newton approximation was used to define the Hessian of the Lagrangian, and the problem was solved via sequential quadratic programming, with each (condensed) quadratic program solved using the dual active set method implemented by the qpOASES solver [60]. On the other hand, the DRL controller was written as ROS integrated Python 3 program. Through ROS communication topics, the DRL controller received both the UR5 joint positions and the references from the arm tracker system on the human operator. The experimental procedure was approved by the Nazarbayev University Institutional Research Ethics Committee (NU-IREC).

Offline reference tracking: execution time and performance

As a first result, we show the data summarizing 25 experimental trials, each with a duration of 40 s. In order to obtain a fair comparison, the same arm tracking data were provided to both DRL and MPC, by recording the time evolution of the reference position p_d, end then providing it to both control schemes. In none of the 25 experiments collisions were observed, as it was the case for the DRL simulations: this confirms the collision avoidance ability of both schemes.

In terms of execution time, DRL showed an average value of 28 μs and a worst-case value of 386 μs; on the other hand, MPC had an average execution time of 18.3 ms, which increased to 73 ms in the worst case. Since MPC had to solve the FHOCP (38) at each sampling instant, while the DRL controller only had to evaluate the output of a DNN for a specific input, the better performance of DRL on this metric could be expected. The execution time of MPC exceeded the sampling interval of 50 ms in very few cases: when this happened, a time delay larger than the sampling interval was introduced into the system, which could lead to a slight loss of performance. Even if the execution time for MPC was mostly lower than the sampling interval, and thus both DRL and MPC could be executed in the setup, the low execution time of DRL would allow one to execute it on less expensive hardware, and/or together with other (concurrent) processes.

In terms of cumulative reward, we first compared the two methods on the average value of J_n for the 25 experiments. As one can notice in Table 2, the DRL controller provided a performance improvement of about 7%; this result was rather consistent through the different experimental trials, since the standard deviation was only about 13% of the mean value of J_n for both cases. The DRL performance was perfectly aligned with that obtained in simulation during the testing phase: the mean value of J_n was approximately the same, although the standard deviation decreased for the experimental data. This provides a strong justification for training the system in simulation,

Motion Planning With Obstacle Avoidance

The methods for robot motion planning with obstacle avoidance used in this work are only a subset of a wider set of approaches, described for instance in [S8]. Most of these methods define a path in the *configuration space* $Q \subseteq \mathbb{R}^{n_q}$, which, in the case study analyzed in this paper, would coincide with the space of manipulator joint angles $q \in \mathbb{R}^6$. The aim is to find a feasible path in the subset of Q that does not contain obstacles (namely, Q_{free}) between a starting configuration q_s and a goal configuration q_g. After a suitable path has been planned, the actual robot motion (that is the evolution of the state variables in time) has to be determined, for example using methods based on spline functions [S9, Ch. 4].

Shared control can be seen as a particular motion planning problem, in which q_g changes with time. The deep reinforcement learning and model predictive control methods considered in this work can be seen as optimal control approaches, since the robot motion is determined based on the maximization of a utility function subject to constraints. A brief non-exhaustive overview of alternative motion planning methods in the presence of obstacles is provided in the following.

VISIBILITY GRAPHS

Visibility graphs [S10] can be used when all obstacles are modeled as polygons and n_q is relatively small. A graph is defined that has all vertices of the polygonal obstacles, together with the starting and goal configurations, as nodes, and, as edges, all straight lines connecting the configurations corresponding to the nodes without crossing any obstacles. A search algorithm such as A* is used to find the shortest path between q_s and q_g, which corresponds to the actual shortest path in Q.

GRID-BASED METHODS

As an alternative, one can ignore the obstacle boundaries and discretize Q_{free} with a uniform grid, obtaining a so-called *grid-based method* [S11, Sec. 10.4]. In this case, one would still obtain a graph, with, as nodes, all vertices of the grid, and, as edges, all straight lines connecting the configurations corresponding to nearby nodes. As in the case of visibility graphs, search methods such as A* can be used to find the shortest path on the graph between the nodes whose configurations are the closest to q_s and q_g. The determined path in Q will be the shortest one for the used level of discretization. This method still suffers from the curse of dimensionality, as the number of nodes explodes as n_q increases.

SAMPLING-BASED METHODS

In order to be able to explore higher-dimensional configuration spaces, sampling-based methods constitute the current state of the art for motion planning [57, Ch. 5]. A tree is built, which is a special type of graph that does not require the use of search algorithms to find the (unique) path between any two nodes. By initializing the tree with a single node corresponding to q_s, these methods are constituted by (i) a function (named *sampler*) that chooses a sample q_σ from Q, (ii) another function that finds the nearest configuration q_n (corresponding to an existing tree node) to q_σ, and (iii) a local planner that adds a new configuration as tree node between q_σ and q_n, and determines a path between q_n and the newly introduced configuration using motion primitives. In case q_s is randomly sampled, this algorithm is referred to as *rapidly-exploring random trees* (RRT), while, if q_s is deterministically chosen using a progressively finer grid, we speak of *rapidly-exploring dense trees* (RDT). A popular variant of RRT, in which the tree is modified ("rewired") after each iteration to avoid convoluted paths, is referred to as RRT*. The solution provided by RRT* asymptotically approaches the actual shortest path in Q as the number of samples increases.

VIRTUAL POTENTIAL FIELD METHODS

A method that directly determines the robot motion, rather than first generating a path, is the one given by *potential field methods* [S13]. This approach, inspired by potential energy fields in the physical world, assigns a low virtual potential to q_g and high virtual potential to the configurations corresponding to obstacles. The direction in Q in which the robot should move is obtained at each sampling instant based on the virtual force vector resulting from the virtual potential field. Following this direction, the robot configuration gets repelled by the obstacles and attracted by q_s, thus eventually converging to q_s. The speed of the robot in Q can be determined based on a simple rule, for example maintaining the speed constant when far from q_g, and gradually decreasing it in a given neighborhood of q_g. This method suffers from the problem of local minima, which might lead the robot to stop at a configuration other than q_g. On the positive side, compared to the motion planning methods described above, virtual potential fields are very suitable for case studies in which the goal point position changes continuously, such as the case study considered in this paper. However, variants of the other methods, especially in the case of sampling-based methods, have also been studied in the case of time-varying environments [S14].

REFERENCES

[S8] J. C. Latombe. *Robot motion planning*. Springer Science & Business Media, Berlin/Heidelberg, Germany, 2012.

[S9] B. Siciliano, L. Sciavicco, L. Villani, Luigi and G. Oriolo. *Robotics: modelling, planning and control*. Springer Science & Business Media, Berlin/Heidelberg, Germany, 2010.

[S10] J. A. Janet, R. C. Luo and M. G. Kay, "The essential visibility graph: An approach to global motion planning for autonomous mobile robots," in *IEEE International Conference on Robotics and Automation (ICRA)*, (Nagoya, Aichi, Japan), pp. 1958–1963, 1995.

[S11] K. M. Lynch, F. C. Park *Modern Robotics*. Cambridge University Press, Cambridge, UK, 2017.

[S12] S. M. Lavalle. *Planning Algorithms*. Cambridge University Press, Cambridge, UK, 2006.

[S13] S. S. Ge and Y. J. Cui. "Dynamic motion planning for mobile robots using potential field method," *Autonomous Robots*, vol. 13, no. 3, pp. 207–222, 2002.

[S14] J. Qi, H. Yang and H. Sun. "MOD-RRT*: a sampling-based algorithm for robot path planning in dynamic environment," *IEEE Transactions on Industrial Electronics*, vol. 68, no. 8, pp. 7244–7251, 2020.

DRL presents a better performance than MPC, but only if the provided reference falls within the distribution of the DRL algorithm policy

TABLE 2 Performance of deep reinforcement learning (DRL) and model predictive control (MPC) algorithms for 25 experiments with a duration of 40 s: J_n (left) and $\hat{J}_n$ (right).

	J_n ($\times 10^4$)		$\hat{J}_n$ ($\times 10^4$)	
	mean	std. dev.	mean	std. dev.
DRL	−1.70	0.22	−2.24	0.32
MPC	−1.84	0.25	−2.48	0.32

since the CoppeliaSim environment could reproduce the actual robot behavior, at least in terms of cumulative cost.

We might argue that this is not a fair comparison, since the MPC strategy was not maximizing exactly the same reward function of DRL: in particular, a purely quadratic function was defined for $\hat{R}_t$ in (37), while a Huber loss function is defined to obtain R_n in (30). Therefore, the equivalent of the a-posteriori evaluated cumulative cost J_n was defined for the MPC reward function, as

$$\hat{J}_n \triangleq \sum_{t=1}^{n} \hat{r}_t \,, \tag{40}$$

and the two algorithms were compared on this metric as well. Also in this case, DRL consistently outperformed MPC of about 9.7%, as one can see again in Table 2.

Real-time reference tracking under different conditions

In order to investigate the reasons for the better performance of DRL as compared to MPC, it is necessary to analyze the time evolution of the system variables. Three sets of experiments, each made of 10 trials of the duration of 40 s each for both controllers, were conducted: in Set 1, the DRL algorithm was executed in real time on the experimental setup, and the reference signal recorded for this first experiment was used with the MPC algorithm for comparison. Screenshots of this type of setting are shown in Figure 5, in which one can see that, for both DRL (on the left) and MPC (on the right), the robot moves up to avoid the box, even if the operator's hand moves approximately horizontally. Conversely, in the second and third set of experiments, the MPC algorithm was executed with a real-time reference from the arm tracker, and the same signal was later provided as reference to the DRL algorithm. In general, executing a control algorithm in real time can be more challenging than using a pre-recorded reference, due to the presence of possible communication delays. In this case, however, the communication delays were negligible, and the results of the experiments with both DRL and MPC executed in real time are shown to prove the actual implementability of the proposed methods.

TABLE 3 Performance of deep reinforcement learning (DRL) and model predictive control (MPC) algorithms for real-time experiments with a duration of 40 s and with reference mutually recorded (Sets 1, 2, and 3): J_n (left) and $\hat{J}_n$ (right).

		J_n ($\times 10^4$)	$\hat{J}_n$ ($\times 10^4$)
Set 1, training	DRL real-time	**−1.91**	**−2.54**
	MPC recorded	−2.13	−2.85
Set 2, higher speed	DRL recorded	**−3.91**	**−5.40**
	MPC real-time	−4.16	−6.17
Set 3, higher z in p_d	DRL recorded	−2.70	−4.25
	MPC real-time	**−1.72**	**−2.23**

We now analyze the system performance for the three sets of experiments. In Set 1, the variations of p_d in time were in the same range observed for the training set. The overall performance of the two controllers is shown in the upper portion of Table 3. These results are in line with those of Table 2: in particular, all costs in Table 3 are about one standard deviation above the corresponding costs of Table 2. The evolution of the reference joint speeds $\dot{q}_d$ for the two control schemes is reported in Figure 6, where one can see that the amplitude of the corresponding control inputs for DRL and MPC were close to each other: this shows that the tuning of the two controllers led to a similar trade-off between control energy and tracking error, thus making their comparison in terms of performance meaningful. The reference tracking for DRL and MPC is reported in Figure 7, in which one can see that both controllers also succeeded to avoid the obstacles (table and box) in the workspace, which would be repeatedly hit if the reference trajectory were to be perfectly tracked. One can see that the quality of the tracking for the x-component of p_d was similar for both controllers. On the other hand, for the y-component of p_d, DRL showed better results; this can be possibly explained by the fact that the human arm motion presented a relatively regular pattern in terms of left-right and right-left movements, which happened along the y axis: this regularity could be exploited by DRL, which learned how to anticipate the operator's motion based on training data. Finally, MPC showed better results on the z-component of p_d, and this was probably due to fact that MPC possessed an explicit, although conservative, formulation of the volume occupied by obstacles (that is,

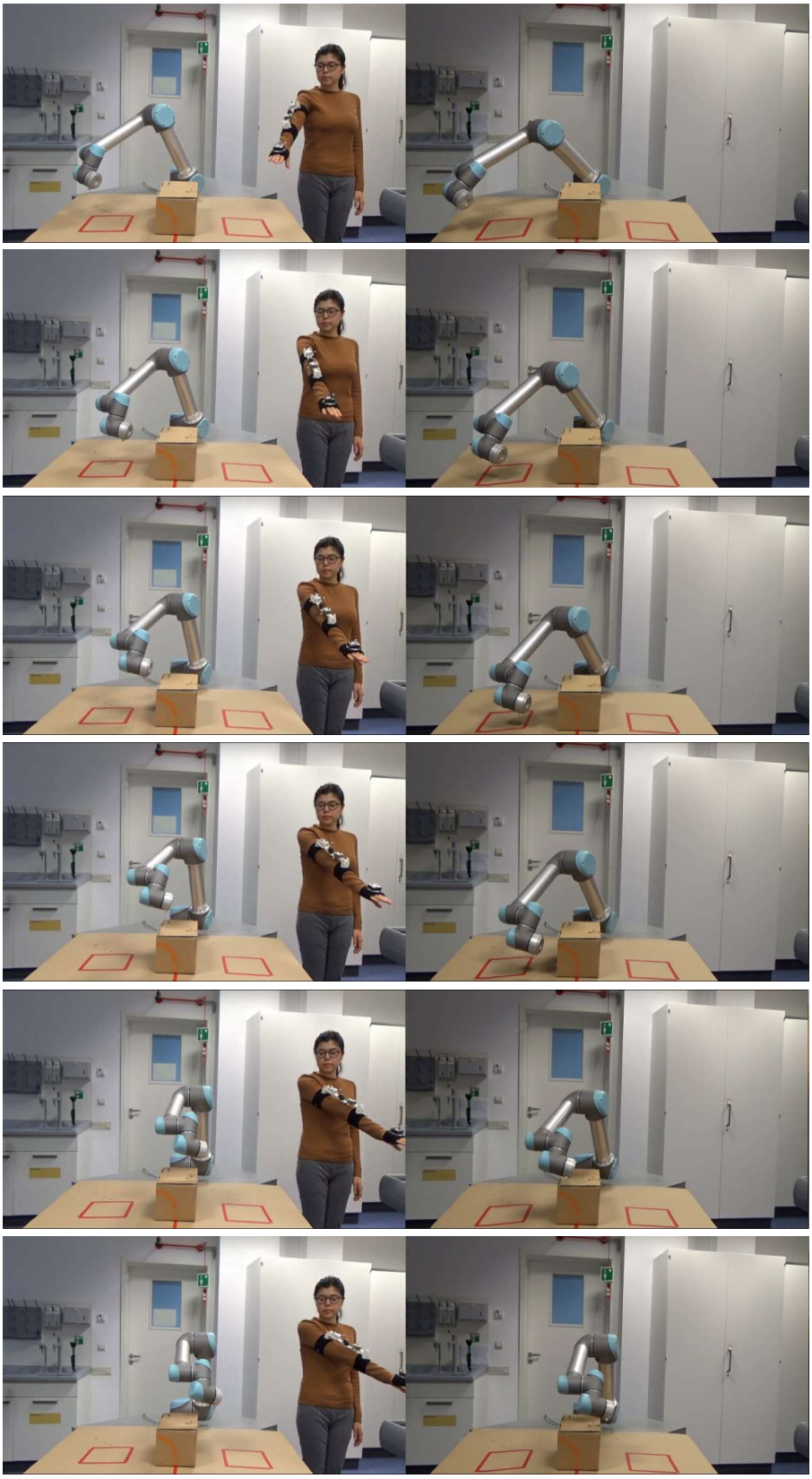

FIGURE 5 Sequence of 6 screenshots corresponding to time instants (from above to below) for Set 1, in which deep reinforcement learning (DRL, left) and model predictive control (MPC, right) follow the same reference trajectory.

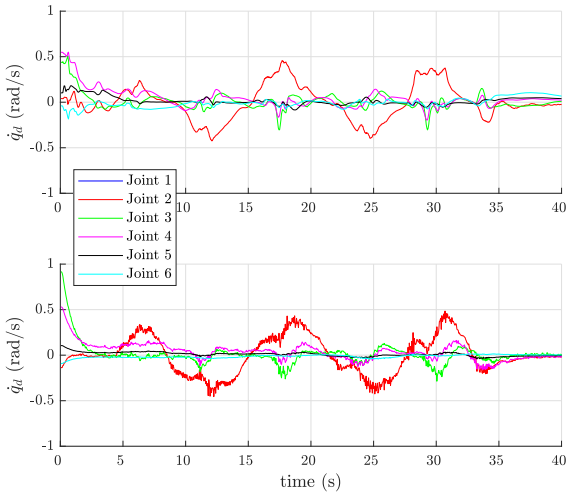

FIGURE 6 Control inputs for DRL (upper plot) and MPC (lower plot). Time evolution of the components of joint angular speed reference vector $\dot{q}_d$ for Set 1.

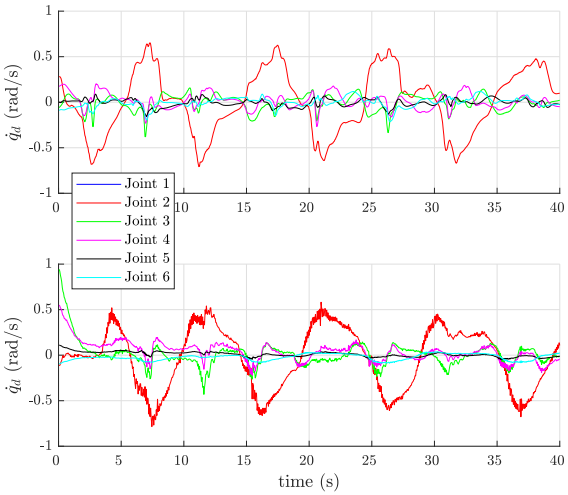

FIGURE 8 Control inputs for DRL (upper plot) and MPC (lower plot). Time evolution of the components of joint angular speed reference vector $\dot{q}_d$ for Set 2.

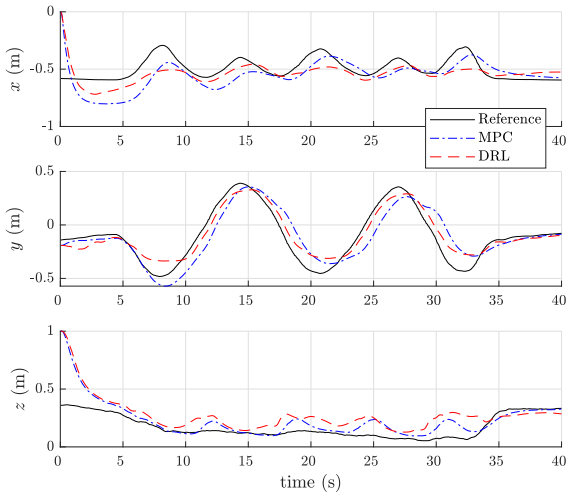

FIGURE 7 End-effector position. Time evolution of the xyz components of reference p_d, and corresponding values of p_e for deep reinforcement learning (DRL) and model predictive control (MPC), in Set 1.

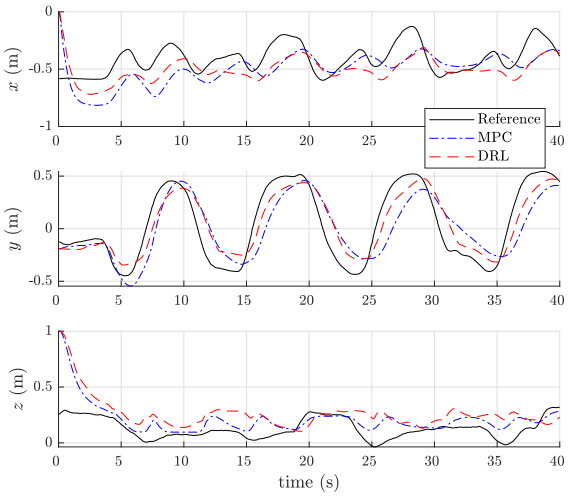

FIGURE 9 End-effector position. Time evolution of the xyz components of reference p_d, and corresponding values of p_e for deep reinforcement learning (DRL) and model predictive control (MPC), in Set 2.

the ellipsoidal approximation of the box allowed the robot to move very close to its edges, but kept the robot up to 4-5 cm farther when situated above the box center), while DRL avoided them based on training data. It is also worth remembering that DRL was trained on a non-conservative model of the obstacles, but accounted for the obstacles using the penalty term r_O defined in (29). As a consequence, $-c_3 r_O$ showed a non-negligible absolute value as compared to that of $-c_1 r_T - c_2 r_U$ already when the robot had a distance of 2-3 cm from the box, and this could have strengthened the tendency of the DRL-controlled robot to remain further away from the obstacle. A steeper penalty function r_O could not be employed to reduce this conservativity, since, in such case, the DRL policy would not converge during training; this can be seen as an example of the more general problem of non-transparent dependence of the DRL policy convergence on hyperparameters values. For all three components of p_d, one can notice that DRL tended to anticipate the human motion more than MPC, thus resulting in a faster control action: this could be due to two factors, namely the precise dynamics model of the UR5 used during training in CoppeliaSim, and the better ability to implicitly forecast the human motion, acquired during training.

In the second and third sets set of experiments, reference trajectories with different characteristics as compared to those used during training were provided, in order to assess the generalization capability of DRL (remembering

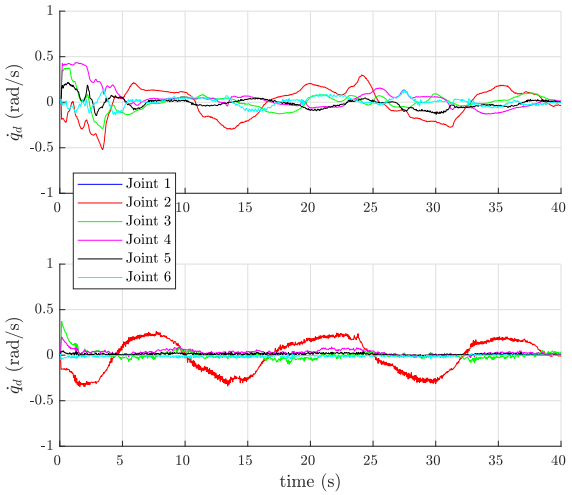

FIGURE 10 Control inputs for DRL (upper plot) and MPC (lower plot). Time evolution of the components of joint angular speed reference vector $\dot{q}_d$ for Set 3.

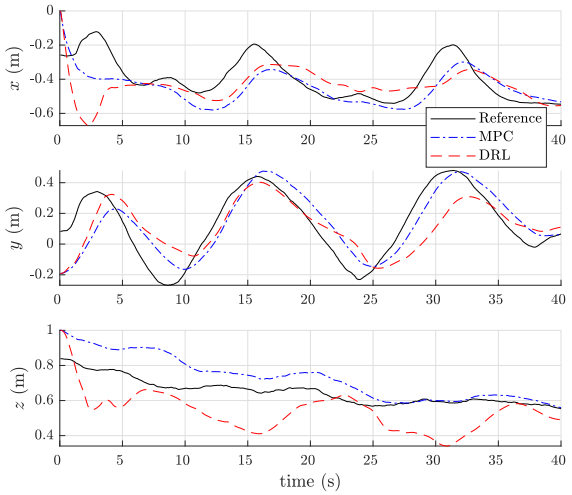

FIGURE 11 End-effector position. Time evolution of the xyz components of reference p_d, and corresponding values of p_e for deep reinforcement learning (DRL) and model predictive control (MPC), in Set 3.

that MPC, being purely model-based, had no need for training, and thus did not suffer from related drawbacks). In particular, in Set 2, a reference was provided with variations of p_d in time that were faster, on average, than those present in the training set. The overall performance is shown in the second portion of Table 3: even though the values of all costs were about twice those of Set 1, DRL still showed better results than MPC. These higher absolute values of the negative cumulative rewards were expected: since the reward penalizes high joint speeds and large tracking errors, it is obvious that a faster reference would both require higher joint speeds, and (with the same tuning) would result in larger tracking errors. The corresponding evolution of the joint speeds, which are again in the same range for DRL and MPC (but are on average with larger absolute values compared to those in Figure 6), can be seen in Figure 8. Also, the reference tracking for both controllers is shown in Figure 9 to which, in spite of the faster trajectories, the same comments apply as for Figure 7.

In Set 3 we provided references p_d in which the z component was considerably larger than the values provided during training. The overall performance is shown in the lower portion of Table 3: even if the variation of p_d in time was similar to that of the training set, and the range of motion of p_d along the y axis was reduced (which led to the lower joint speeds observable in Figure 10), DRL exhibited a considerable performance degradation, with a 41% and 67% increase of the average magnitudes of J_n and $\hat{J}_n$, respectively, as compared to Set 1. This is due to the fact that the provided reference trajectories were outside the distribution of the reinforcement learning algorithm policy, and the results are visible in Figure 11, in which one can see that the tracking error in the z axis is of considerable entity. On the other hand, MPC did not show any substantial performance degradation, thanks to its model-based nature. By taking advantage of the smaller range of motion of p_d along the y axis as compared to Set 1, MPC even managed to decrease the absolute value of both cumulative reward functions. A single frame of the sequence of motion represented in Figure 11 can be seen in Figure 12, where one can observe the large discrepancy in terms of z for the two controllers.

To complement these results and provide better understanding, videos from all three sets of experiments are available at https://youtu.be/5jWXP9ozgfk.

COMPARISON WITH PREVIOUS WORKS

In [49], DRL outperformed MPC in a distributed aircraft control framework for wildfire surveillance, due to two main reasons. First, each aircraft was controlled in a decentralized fashion, because an MPC scheme with a higher degree of coordination would have required to increase the suboptimality of the solution (due to real-time computational constraints), with a consequent loss of performance; instead, the proposed DRL schemes provided a higher level of coordination, thanks to their lower real-time computational complexity. The second reason was also related to computational feasibility, since the prediction horizon of MPC had to be maintained relatively short, which gave it a reduced ability to plan the aircraft motion: this resulted in sharper turns when tracking the wildfire front, as compared to DRL. These drawbacks of MPC compared to DRL were not found in our work, since we used a centralized scheme for both DRL and MPC, and the prediction horizon of MPC was sufficiently long to avoid sudden changes in the robot motion for avoiding obstacles.

FIGURE 12 Screenshot from the video representing a single time instant in Set 3, in which deep reinforcement learning (DRL, left) and model predictive control (MPC, right) follow the same reference trajectory.

In [50], a car was controlled to execute a safe and comfortable merging maneuver into dense traffic. The proposed DRL approach was compared against several MPC strategies, which differed for the method used to predict the motion of the other cars (no motion, constant speed, motion learned via a recurrent neural network). The MPC strategies based on no-motion or constant-speed predictions suffered from the inability to account for inter-vehicle interactions, which led to a lower success rate as compared to DRL and to the MPC strategy with motion of the other cars learned via recurrent neural network (even if, for the latter, a more relaxed criterion was used to assess the success rate). DRL outperformed all other methods in terms of lowering the time to merge into the traffic, and was less affected by the distribution of drivers on the road than the MPC strategies. Also in this case, there is no obvious overlapping of conclusions with our work, also due to the very different application.

Finally, [51] compared DRL and MPC on an adaptive cruise control application. When there were no modeling errors and the testing inputs were within the distribution of the DRL algorithm policy, DRL and MPC performed similarly, as long as MPC had a sufficiently long prediction horizon. The DRL performance worsened when the testing inputs were outside the distribution of the DRL algorithm policy. On the other hand, when modeling errors were inserted (for instance, by increasing control delay or disturbances) DRL performed better than MPC. The authors could not provide a formal explanation for this; however, they gave the following intuitive explanation: the DRL "state transition is based on expectation of probabilities although the state-action mapping is deterministic. The probabilistic state transition allows for environment stochasticity that can be represented as modeling errors." The results in [51] had in common with our work that the DRL performance was affected by inputs to the DNN outside the distribution of the DRL policy.

The main difference of our work as compared to [49]–[51] mainly consists of the implementing DRL and comparing it with MPC, analyzing pros and cons in a different field of application (industrial robotics rather than aircraft or autonomous vehicle control), for a different type of control problem (shared control rather than control of autonomous agents), and via experiments rather than simulations.

CONCLUSIONS

The proposed DRL strategy for manipulator teleoperation succeeded in avoiding obstacles while aiming at tracking the provided reference position. The observed advantages with respect to MPC can be summarized as a drastic (and largely expected) reduction in terms of execution time, and a satisfactory improvement in terms of performance, measured via the cost functions defined for DRL and MPC. On the other hand, DRL failed to provide acceptable results when the reference was outside the range used during training: this was also expected, since the data-driven nature of DRL requires training data that cover the whole range of possible system operation.

ACKNOWLEDGMENTS

This work was supported in part by Nazarbayev University under Collaborative Research Project no. 091019CRP2118, in part by the Kazakhstan Ministry of Education and Science under Young Researchers Grant Project No. AP08052091, and in part by the Italian Ministry for Research in the Framework of the 2017 Program for Research Projects of National Interest (PRIN) under Grant 2017YKXYXJ.

AUTHOR INFORMATION

Matteo Rubagotti received the Ph.D. degree in Electronics, Computer Science, and Electrical Engineering from the University of Pavia, Pavia, Italy, in 2010. Since 2018, he has been an Associate Professor of Robotics and Mechatronics at Nazarbayev University, Nur-Sultan, Kazakhstan, where he directs the Robot Control and Learning lab. Previously to his post at Nazarbayev University, he was a Postdoctoral Fellow first at the University of Trento, Trento, Italy, and then at IMT Institute for Advanced Studies, Lucca, Italy, and later a Lecturer in Control Engineering at the University of Leicester, Leicester, UK. His current research interests include numerical optimal control, model predictive control and sliding mode control, and their application

to robotics and mechatronics. Dr. Rubagotti is Subject Editor of the International Journal of Robust and Nonlinear Control, and member of the conference editorial boards of the IEEE Control System Society and of the European Control Association. He is IEEE Senior Member.

Bianca Sangiovanni received the Bachelor degree (2015) and Master degree (2017) in Computer Engineering from the University of Pavia, Pavia, Italy. From November 2017 to December 2020, she was a Ph.D. student at the Identification and Control of Dynamic Systems (ICDS) laboratory at the University of Pavia, receiving her Ph.D. in 2021. Her main research interests are focused on robot control, automation and deep reinforcement learning. She has been a Student Member of the IEEE Control System Society since 2020.

Aigerim Nurbayeva received the B.Sc. in Radio Engineering, Electronics and Telecommunications from L. M. Gumilov Eurasian National University, Nur-Sultan, Kazakhstan (2017) and the M.Sc. in Robotics from Nazarbayev University, Nur-Sultan, Kazakhstan (2020). She is currently a Ph.D. candidate in Robotics Engineering at Nazarbayev University. Her research interests focus on nonlinear model predictive control, motion tracking systems, neural networks, and deep reinforcement learning.

Gian Paolo Incremona is Assistant Professor of Automatic Control at Politecnico di Milano, Italy. He was a student of the Almo Collegio Borromeo of Pavia, and of the class of Science and Technology of the Institute for Advanced Studies IUSS of Pavia. He received the Bachelor and Master degrees (with highest honor) in Electric Engineering, and the Ph.D. degree in Electronics, Electric and Computer Engineering from the University of Pavia in 2010, 2012 and 2016, respectively. From October to December 2014, he was with the Dynamics and Control Group at the Eindhoven Technology University, The Netherlands. He was a recipient of the 2018 Best Young Author Paper Award from the Italian Chapter of the *IEEE Control Systems Society*, and he has been a member of the conference editorial boards of the *IEEE Control System Society* and of the *European Control Association*, since 2018. At present, he is *Associate Editor* of the journal *Nonlinear Analysis: Hybrid Systems* and of *International Journal of Control*. His research interests include variable structure control, model predictive control, networked control, industrial robotics, power systems and glycemia control in diabetic subjects.

Antonella Ferrara received the M.Sc. degree in electronic engineering and the Ph.D. degree in computer science and electronics from the University of Genoa, Italy, in 1987 and 1992, respectively. Since 2005, she has been Full Professor of automatic control at the University of Pavia, Italy. Her main research interests include sliding mode control and nonlinear control applied to automotive systems, power networks, and robotics, as well as road traffic control. She is author and co-author of more than 400 publications including more than 140 journal papers, 2 monographs and one edited book. At present, she is Associate Editor of Automatica and of the International Journal of Robust and Nonlinear Control, as well as Senior Editor of IEEE Transactions on Intelligent Vehicles. She was Associate Editor of the IEEE Transactions on Control Systems Technology, IEEE Transactions on Automatic Control and IEEE Control Systems Magazine. Dr. Ferrara is the EUCA Conference Editorial Board Chair. She is a member of the IEEE TC on Automotive Control, IEEE TC on Smart Cities, IEEE TC on Variable Structure Systems, IFAC Technical Committee on Nonlinear Control Systems, IFAC TC on Transportation Systems, and IFAC Technical Committee on Intelligent Autonomous Vehicles. She is a Fellow of IEEE.

Almas Shintemirov received the engineer's degree in electrical engineering and the Ph.D. degree in technical sciences from Pavlodar State University, Pavlodar, Kazakhstan, in 2001 and 2004, respectively, and the Ph.D. degree in electrical engineering and electronics from the University of Liverpool, Liverpool, U.K., in 2009. In 2009 he joined Nazarbayev University, where in 2011 he became a founding faculty member of the Department of Robotics and Mechatronics. He is currently Associate Professor of Robotics and Mechatronics at Nazarbayev University, where he directs the Astana Laboratory for Robotic and Intelligent Systems (www.alaris.kz) and where, since 2019, he has also served as Acting Chair of the Department of Electrical and Computer Engineering. His research interests include assistive robotics, mobile and industrial robotics, robot optimal control, mechatronic systems design, and computational intelligence. For his academic achievements, he received the Certificate of Merit of the Republic of Kazakhstan (state award) in 2017 and the Kazakhstan Scopus Award – Top Researcher in Engineering and Technologies - from Elsevier in 2018. He is IEEE Senior Member.

REFERENCES

[1] J. Vertut and P. Coiffet, *Teleoperations and robotics: evolution and development*. Prentice-Hall, Inc., 1986.

[2] W.-K. Yoon, T. Goshozono, H. Kawabe, M. Kinami, Y. Tsumaki, M. Uchiyama, M. Oda, and T. Doi, "Model-based space robot teleoperation of ets-vii manipulator," *IEEE Transactions on Robotics and Automation*, vol. 20, no. 3, pp. 602–612, 2004.

[3] A. Flores-Abad, O. Ma, K. Pham, and S. Ulrich, "A review of space robotics technologies for on-orbit servicing," *Progress in Aerospace Sciences*, vol. 68, pp. 1–26, 2014.

[4] J. Funda, R. H. Taylor, B. Eldridge, S. Gomory, and K. G. Gruben, "Constrained cartesian motion control for teleoperated surgical robots," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 3, pp. 453–465, 1996.

[5] R. H. Taylor, A. Menciassi, G. Fichtinger, P. Fiorini, and P. Dario, "Medical robotics and computer-integrated surgery," in *Springer handbook of robotics*, 2016, pp. 1657–1684.

[6] C. Passenberg, A. Peer, and M. Buss, "A survey of environment, operator, and task-adapted controllers for teleoperation systems," *Mechatronics*, vol. 20, no. 7, pp. 787–801, 2010.

[7] P. F. Hokayem and M. W. Spong, "Bilateral teleoperation: An historical survey," *Automatica*, vol. 42, no. 12, pp. 2035–2057, 2006.

[8] H. Boessenkool, D. A. Abbink, C. J. M. Heemskerk, F. C. T. van der Helm, and J. G. W. Wildenbeest, "A task-specific analysis of the benefit of haptic shared control during telemanipulation," *IEEE Transactions on Haptics*, vol. 6, no. 1, pp. 2–12, 2012.

[9] C. Pacchierotti, L. Meli, F. Chinello, M. Malvezzi, and D. Prattichizzo, "Cutaneous haptic feedback to ensure the stability of robotic teleoperation systems," *The International Journal of Robotics Research*, vol. 34, no. 14, pp. 1773–1787, 2015.

[10] S. Musić and S. Hirche, "Control sharing in human-robot team interaction," *Annual Reviews in Control*, vol. 44, pp. 342–354, 2017.

[11] D. Lee, A. Franchi, H. I. Son, C. Ha, H. H. Bülthoff, and P. R. Giordano, "Semiautonomous haptic teleoperation control architecture of multiple unmanned aerial vehicles," *IEEE/ASME Transactions on Mechatronics*, vol. 18, no. 4, pp. 1334–1345, 2013.

[12] S.-Y. Jiang, C.-Y. Lin, K.-T. Huang, and K.-T. Song, "Shared control design of a walking-assistant robot," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 6, pp. 2143–2150, 2017.

[13] X. Deng, Z. L. Yu, C. Lin, Z. Gu, and Y. Li, "A bayesian shared control approach for wheelchair robot with brain machine interface," *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 28, no. 1, pp. 328–338, 2019.

[14] J. Luo, Z. Lin, Y. Li, and C. Yang, "A teleoperation framework for mobile robots based on shared control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 377–384, 2019.

[15] D. Rakita, B. Mutlu, M. Gleicher, and L. M. Hiatt, "Shared control–based bimanual robot manipulation," *Science Robotics*, vol. 4, no. 30, 2019.

[16] M. Selvaggio, M. Cognetti, S. Nikolaidis, S. Ivaldi, and B. Siciliano, "Autonomy in physical human-robot interaction: a brief survey," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 17 989–7996, 2021.

[17] C. J. Payne, K. Vyas, D. Bautista-Salinas, D. Zhang, H. J. Marcus, and G. Z. Yang, "Shared-control robots," in *Neurosurgical Robotics*, 2021, pp. 63–79.

[18] Q. Zhu, T. Zhou, and J. Du, "Upper-body haptic system for snake robot teleoperation in pipelines," *Advanced Engineering Informatics*, vol. 51, p. 101532, 2022.

[19] A. Hansson and M. Servin, "Semi-autonomous shared control of large-scale manipulator arms," *Control Engineering Practice*, vol. 18, no. 9, pp. 1069–1076, 2010.

[20] G. Brantner and O. Khatib, "Controlling ocean one: Human–robot collaboration for deep-sea manipulation," *Journal of Field Robotics*, vol. 38, no. 1, pp. 28–51, 2021.

[21] M. Selvaggio, F. Abi-Farraj, C. Pacchierotti, P. R. Giordano, and B. Siciliano, "Haptic-based shared-control methods for a dual-arm system," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4249–4256, 2018.

[22] S. Parsa, D. Kamale, S. Mghames, K. Nazari, T. Pardi, A. R. Srinivasan, G. Neumann, M. Hanheide, and G. E. Amir, "Haptic-guided shared control grasping: collision-free manipulation," in *Proc. IEEE International Conference on Automation Science and Engineering*, Hong Kong, China, 2020, pp. 1552–1557.

[23] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[24] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An introduction to deep reinforcement learning," *Foundations and Trends® in Machine Learning*, vol. 11, no. 3-4, pp. 219–354, 2018.

[25] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proc. of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, New Orleans, LA, USA, 2018.

[26] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Marina Bay Sands, Singapore, 2017, pp. 3389–3396.

[27] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke *et al.*, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *Proc. Conference on Robot Learning*, Zurich,Switzerland, 2018, pp. 651–673.

[28] Y. Tsurumine, Y. Cui, E. Uchibe, and T. Matsubara, "Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation," *Robotics and Autonomous Systems*, vol. 112, pp. 72–83, 2019.

[29] B. Thananjeyan, A. Balakrishna, U. Rosolia, F. Li, R. McAllister, J. E. Gonzalez, S. Levine, F. Borrelli, and K. Goldberg, "Safety augmented value estimation from demonstrations (SAVED): Safe deep model-based RL for sparse cost robotic tasks," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3612–3619, 2020.

[30] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov, and S. Levine, "Learning complex dexterous manipulation with deep reinforcement learning and demonstrations," in *Proc. Robotics: Science and Systems (RSS)*, Pittsburgh, Pennsylvania, USA, 2018.

[31] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei, "Surreal: Open-source reinforcement learning framework and robot manipulation benchmark," in *Proc. Conference on Robot Learning*, Zurich, Switzerland, 2018, pp. 767–782.

[32] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, 2018, pp. 6244–6251.

[33] B. Sangiovanni, A. Rendiniello, G. P. Incremona, A. Ferrara, and M. Piastra, "Deep reinforcement learning for collision avoidance of robotic manipulators," in *Proc. European Control Conference*, Jul. 2018.

[34] B. Sangiovanni, G. P. Incremona, M. Piastra, and A. Ferrara, "Self-configuring robot path planning with obstacle avoidance via deep reinforcement learning," *IEEE Control Systems Letters*, vol. 5, no. 2, pp. 397–402, 2021.

[35] W. Yuan, J. A. Stork, D. Kragic, M. Y. Wang, and K. Hang, "Rearrangement with nonprehensile manipulation using deep reinforcement learning," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia, 2018.

[36] E. F. Camacho and C. B. Alba, *Model predictive control*. Springer science & business media, 2013.

[37] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model Predictive Co: Theory, Computation, and Design*, 2nd ed. Nob-Hill Publishing, 2017.

[38] L. Grüne and J. Pannek, "Nonlinear model predictive control," in *Nonlinear model predictive control*. Springer, 2017.

[39] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit - an open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[40] J. Mattingley and S. Boyd, "Cvxgen: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, no. 1, pp. 1–27, 2012.

[41] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos, "A simple and efficient algorithm for nonlinear model predictive control," in *Proc. IEEE Conference on Decision and Control*, Melbourne, Australia, 2017, pp. 1939–1944.

[42] J. Sheng and M. W. Spong, "Model predictive control for bilateral teleoperation systems with time delays," in *Canadian Conference on Electrical and Computer Engineering*, Halifax, NS, Canada, 2004, pp. 1877–1880.

[43] S. Sirouspour and A. Shahdi, "Model predictive control for transparent teleoperation under communication time delay," *IEEE Transactions on Robotics*, vol. 22, no. 6, pp. 1131–1145, 2006.

[44] T. Faulwasser, T. Weber, P. Zometa, and R. Findeisen, "Implementation of nonlinear model predictive path-following control for an industrial robot," *IEEE Transactions on Control Systems Technology*, vol. 25, no. 4, pp. 1505–1511, 2016.

[45] G. B. Avanzini, A. M. Zanchettin, and P. Rocco, "Constrained model predictive control for mobile robotic manipulators," *Robotica*, vol. 36, no. 1, pp. 19–38, 2018.

[46] M. Krämer, C. Rösmann, F. Hoffmann, and T. Bertram, "Model predictive control of a collaborative manipulator considering dynamic obstacles," *Optimal Control Applications and Methods*, vol. 41, pp. 1211–1232, 2020.

[47] S. Gu, T. P. Lillicrap, I. Sutskever, and S. Levine, "Continuous deep Q-learning with model-based acceleration," in *Proc. International Conference on Machine Learning*, New York City, NY, USA, 2016.

[48] M. Rubagotti, T. Taunyazov, B. Omarali, and A. Shintemirov, "Semi-autonomous robot teleoperation with obstacle avoidance via model predictive control," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2746–

2753, 2019.

[49] K. D. Julian and M. J. Kochenderfer, "Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 8, pp. 1768–1778, 2019.

[50] D. M. Saxena, S. Bae, A. Nakhaei, K. Fujimura, and M. Likhachev, "Driving in dense traffic with model-free reinforcement learning," in *Proc. International Conference on Robotics and Automation*, 2020, pp. 5385–5392.

[51] Y. Lin, J. McPhee, and N. L. Azad, "Comparison of deep reinforcement learning and model predictive control for adaptive cruise control," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 2, pp. 221–231, 2021.

[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[53] C. Possieri, G. P. Incremona, G. C. Calafiore, and A. Ferrara, "An iterative data-driven linear quadratic method to solve nonlinear discrete-time tracking problems," *IEEE Transactions on Automatic Control*, vol. 66, no. 11, pp. 5514–5521, 2021.

[54] C. Ericson, *Real-time collision detection*. CRC Press, 2004.

[55] A. Shintemirov, T. Taunyazov, B. Omarali, A. Nurbayeva, A. Kim, A. Bukeyev, and M. Rubagotti, "An open-source 7-DOF wireless human arm motion-tracking system for use in robotics research," *Sensors*, vol. 20, no. 11, p. 3082, 2020.

[56] S. James, M. Freese, and A. J. Davison, "PyRep: Bringing V-REP to deep robot learning," *arXiv preprint arXiv:1906.11176*, 2019.

[57] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[58] B. Houska, H. J. Ferreau, and M. Diehl, "An auto-generated real-time iteration algorithm for nonlinear mpc in the microsecond range," *Automatica*, vol. 47, no. 10, pp. 2279–2285, 2011.

[59] R. Quirynen, M. Vukov, M. Zanon, and M. Diehl, "Autogenerating microsecond solvers for nonlinear mpc: a tutorial using acado integrators," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 685–704, 2015.

[60] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.